

# Hybrid Mobile Applications - ICD0018

TalTech IT College, Andres Käver, 2021-2022, Fall semester

Email: [andres.kaver@taltech.ee](mailto:andres.kaver@taltech.ee)



# React N – Class vs Functional

2

- ▶ React and React Native has two approaches how to write components
- ▶ Class based – classical, full support
- ▶ Functional
  - ▶ Initially limited, no state
  - ▶ **Hooks** were implemented at the start of 2019
  - ▶ Now the recommended way!
  - ▶ Some edge cases still need class-based components (lifecycle events)

# React N – Class based component

3

- ▶ Class based component inherits from `Component<P={}, S={}>`
- ▶ Declare interfaces for properties and state
- ▶ Properties – read-only data from parent component
- ▶ State – components internal data, set initial value in constructor

```
export interface Props {  
  label: string;  
}  
  
export interface State {  
  counter: number;  
}
```

```
export class Hello extends Component<Props, State> {  
  constructor(props: Props) {  
    super(props);  
    this.state = { counter: 0 };  
  }  
}
```

# React N - Class based component

4

- ▶ Access both state and properties is provided via inheritance.
  - ▶ `this.props.xxx` and `this.state.xxx`
- ▶ State – modify state with **`this.setState({...})`** function. Only include changed properties as parameters – objects are merged.
- ▶ This will allow React to keep track of state changes and only update required UI components.
- ▶ Never modify state directly!

```
render() {  
  return (  
    <View style={styles.root}>  
      <Button  
        title="+"  
        onPress={() => {  
          this.setState({ counter: this.state.counter + 1 });  
        }}  
      />  
    </View>  
  );  
}
```

# React N – functional components

5

- ▶ Properties – declare interface, and specify props as parameter to function
- ▶ No this in functional components (no object instance here)

```
export interface Props {  
  label: string;  
}  
  
const HelloFn = (props: Props): JSX.Element => (  
  <View style={styles.root}>  
    <Text style={styles.greeting}>{props.label}</Text>  
  </View>  
);
```

# React N – functional componets

6

- ▶ State in functional components – use hooks!
- ▶ A Hook is a kind of function that lets you “hook into” React features.
- ▶ Most common hooks are
  - ▶ `useState`
  - ▶ `useEffect`
  - ▶ `useContext`
  - ▶ `useReducer`

# React N – functional components

7

- ▶ `const [state, setState] = useState(initialState);`
- ▶ Returns a stateful value, and a function to update it.
- ▶ `useState` can be used several times, to create separate state containers.

```
const HelloFn = (props: Props): JSX.Element => {
  const [counter, setCounter] = useState(0);
  return (
    <View style={styles.root}>
      <Button title="-" onPress={() => setCounter(counter - 1)} />
      <Text style={styles.greeting}>
        {props.label} {counter}
      </Text>
      <Button title="+" onPress={() => setCounter(counter + 1)} />
    </View>
  );
};
```

# React N – functional components

8

- ▶ Unlike `setState` in class components, `useState` does not merge update objects. Use object spread syntax.
- ▶ Another option is `useReducer`, which is more suited for managing state objects that contain multiple sub-values.
- ▶ If the new state is computed using the previous state, you can pass a function to `setState`. The function will receive the previous value and return an updated value.



# React N

9

▶ THE END!